Introduction to R

María-Eglée Pérez and Luis Raúl Pericchi

Biostatistics, Epidemiology and Bioinformatics Core (BEBIC)
UPR-MDACC Partnership for Excellence in Cancer Research
NIH NCI U54 GRANT 2U54CA096297-11

Contents

The S Language

- Developed at Bell Laboratories (AT&T) by the end of the 70's and the begining of the 80's by Richard Becker, John Chambers and Allan Wilks
- In the begining the system was designed to be flexible and interactive, specially suitable for exploratory data analysis. Most statistical functionality was added later.
- S is still being developed.

Implementations of *S*: *R*

- *R* was initially developed by Ross Ihaka and Robert Gentleman from the University of Auckland (New Zealand)
- Since 1995 it is distributed freely under the terms of GNU general license
- *R* developement has turned an international collaborative effort of many volunteers
- Since 1997 a "core group" (which includes John Chambers) manages *R* developement.

Characteristics of *R*

R is an integrated set of programs for manipulation of data, calculations and graphics. It has built in

- Effective storage and manipulation of data.
- Operators for calculations over indexed variables (specially matrices)
- A wide, coherente and integrated collection of tools for data analysis.
- Tools for graphical data analysis, both on screen or printed.
- A well developed programming language, simple and effective, including conditionals, cicles, recursion and several posibilities for input and output

Characteristics of *R*

R is an "environment", that is, a totally designed and coherent system, not an incremental collection of specific and inflexible tools.

R can be extended by means of user written programs, or downloading packages from the Internet

Installation of *R*

- Precompiled binary distributions for Windows, MacOS and several distrubutions of Linux can be downloaded from `http://cran.r-project.org` or any of its mirrors.
- For installing *R* in Windows, just double click `.exe` file.
- Source files can be downloaded to install *R* in other operating systems.

Running and Quitting *R*

- For running *R*, just double click the icon on the desktop, or use the "Start" menu.
- For quitting *R*, write
  `>q()`
  (Note that the command for quitting is a function!)
  A window will appear with the question: "Save workspace image?" and three buttons: "Yes", "No" and "Cancel". If "No" is selected, all the objects created during the session will be lost.

Interfaces for R

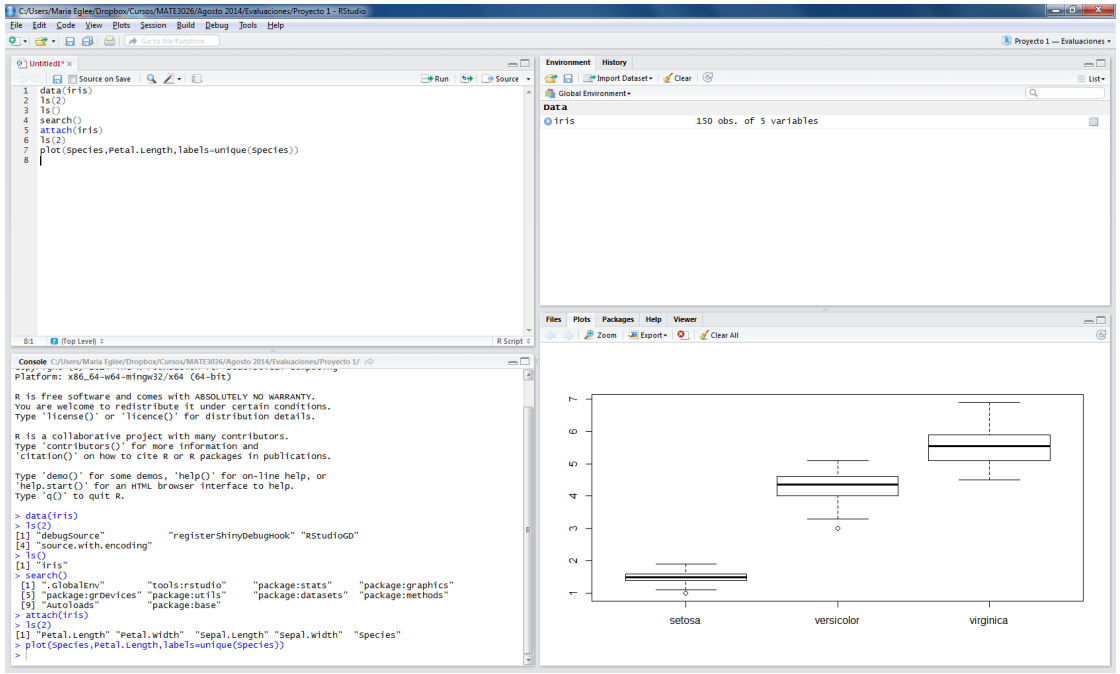
Even though R can be used directly, many users prefer using an interface. Some common interfaces are:

- RStudio: is an "Integrated Development Environment" for R (`http://www.r-studio.com`)
- RCommander: a Graphic User Interface (GUI) for R, which incorporates point and click menus for many statistical procedures. It can be integrated with RStudio (`http://www.rcommander.com`)
- Deducer: another GUI for R (`http://www.deducer.org`).

RStudio Window

Text Editor

Workspace



Console

Graphs

Getting Help

- In R, on-line help can be obtained clicking the "Help" menu at the menu bar.
- HTML help can also be obtained using the command
  `> help.start()`
- Help on an specific command can be also obtained using the command
  `> help(`*command*`)` or `> ?`*command*
  A search by keyword can be done using `??`*keyword*.
- In RStudio, there is a "help" tab in the graph section of the screen.
- Some documentation can be found at `www.r-project.org`. Also, there are a variety of introductory books.

Extending R functionalities: Packages

One of the biggest advantages of R is its wide community of users and collaborators. Members of this community produce extensions to R, or *packages*.

A list of existing packages can be found at `http://cran.r-project.org`. On October 27, 2014, there were 5995 packages!

Packages can be installed:

- with command `install.packages("`*name*`")`;
- using the menu item `Packages > Install packages` in R (R for windows);
- using the tab "packages" in the graph region of the RStudio window;

If we are going to use a package during the actual session, we can load it using `require(`*name*`)` or `library(`*name*`)`

We can see which packages are loaded using `search()` (which returns the search list) or `sessionInfo()`

Basic Mathematical Operations and Mathematical Functions

- Symbols for arithmetical operations are: $+$ (sum), $-$ (difference), $*$ (product), / (quotient), ^ (power).
- Order of evaluation is : power - product or quotient - sum or difference.
- Many mathematical functions are available: `sqrt`, `exp`, `log`, `sin`, `cos`, `tan`, etc. They can be composed and incorporated in calculations.

```
> 7+2*3
[1] 13
> (7+2)*3
[1] 27
> 12/2+4
[1] 10
> 5+2^3
[1] 13
> 4*sqrt(2)
[1] 5.656854
> log(sqrt(2)+1)
[1] 0.8813736
```

Variables and Assignments

- The result of any operation can be assigned to a variable using the operators "=" or "<-".
- The variable will keep the assigned value till it is erased or overwritten by the user.
- *R* is case-sensitive; so, `X` and `x` are different variables.
- Variables can contain characters

```
> x<-5+6^2
> x
[1] 41
> y=8+2
> y
[1] 10
> x=9*sqrt(20)
> x
[1] 40.24922
> name1="Maria"
> name1
[1] "Maria"
```

Listing Workspace and Erasing Variables

- Stored objects can be seen using commands `ls` and `objects`.
- If the result of an operation or function is not stored in a variable, its value is lost.
- Variables can be erased from workspace using `rm`

```
> ls()
[1] "name1" "x" "y"
> objects()
[1] "name1" "x" "y"
> rm(x,y)
> ls()
[1] "name1"
```

Vectors

Usually, we'll need to work with ordered collections of objects (numbers, characters, etc) or *vectors*. Vectors can be created in several ways

- Concatenating objects (Command c)
  It allows to concatenate new or existing objects:

```
> x=c(1,2,3,4,5)
> x
[1] 1 2 3 4 5
> x=c(x,6)
> x
[1] 1 2 3 4 5 6
> y=c(20,40,60)
> y
[1] 20 40 60
```

```
> z=c(x,y)
> z
[1]  1  2  3  4  5  6 20 40 60
> name2="Eglee"
> name3="Perez"
> complete.name=c(name1,name2,name3)
> complete.name
[1] "Maria" "Eglee" "Perez"
```

- Creating sequences (Command `seq`)
  This command generates regular number sequences (equally spaced).
  Its general form is `seq(`*min*, *max*,*[increment]* `)` (*increment* is optional, with a default value of 1).

```
> seq(1,10)
 [1]  1  2  3  4  5  6  7  8  9 10
> seq(1960,2000,5)
[1] 1960 1965 1970 1975 1980 1985 1990 1995 2000
> seq(0,1,.1)
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> seq(1,0,-0.1)
 [1] 1.0 0.9 0.8 0.7 0.6 0.5 0.4 0.3 0.2 0.1 0.0
```

Short commands for sequences with increment 1 can be used.

```
> seq(10)
 [1]  1  2  3  4  5  6  7  8  9 10
> 1:10
 [1]  1  2  3  4  5  6  7  8  9 10
 > 10:1
 [1] 10  9  8  7  6  5  4  3  2  1
```

- Repeating regular patterns (Command `rep`)
  General form: `rep`(*pattern*, *number of repetitions*)

```
> rep(1,10)
 [1] 1 1 1 1 1 1 1 1 1 1
> rep(1:5,2)
 [1] 1 2 3 4 5 1 2 3 4 5
> ind=rep(c("yes","no"),3)
> ind
[1] "yes" "no"  "yes" "no"  "yes" "no"
> logic=rep(c(TRUE,FALSE),5) # vector with logical values
> logic
 [1]  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE
 [8] FALSE  TRUE FALSE
> rep(1:3,1:3)
[1] 1 2 2 3 3 3
> rep(1:3,length=10)
 [1] 1 2 3 1 2 3 1 2 3 1
```

All arithmetical operations and mathematical functions can be applied to vectors. They will be performed for each component.

```
> a=1:10
> b=a+5
> b
 [1]  6  7  8  9 10 11 12 13 14 15
> a+b
 [1]  7  9 11 13 15 17 19 21 23 25
> c=sqrt(a)
> c
 [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490
 [7] 2.645751 2.828427 3.000000 3.162278
> d=10^-a
> d
  [1] 1e-01 1e-02 1e-03 1e-04 1e-05 1e-06 1e-07 1e-08 1e-09
 [10] 1e-10
```

Brackets ([]) allow to extract elements of a vector

```
> c
 [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490
 [7] 2.645751 2.828427 3.000000 3.162278
> c[3]
[1] 1.732051
> c[10]
[1] 3.162278
> c[11]
[1] NA
```

There are many functions wich are specific for vectors. Some of them are:

`length`(v)	Length of a vector
`min`(v)	Minimum
`max`(v)	Maximum
`sum`(v)	Sum of vector components
`mean`(v)	Mean
`var`(v)	Sample variance
`sort`(v)	Orders the vector increasingly
$\vdots$	$\vdots$

Depending on the objects they contain, vectors will have different *classes*

```
> class(c)
[1] "numeric"
> class(ind)
[1] "character"
> class(logic)
[1] "logical"
```

The class of a vector can be changed (with the due consequences!)

```
> as.numeric(logic)
 [1] 1 0 1 0 1 0 1 0 1 0
> as.character(a)
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10"
```

Data Frames

Usually, we have a set of observations for each subjects in the study. A *data frame* is a structure with matrix shape, in which different data columns (numeric, character or logical) can be stored. In general, rows correspond to different subjects, and columns correspond to variables observed for each subject.
Names can be assigned to the rows and columns of a data frame.
As an example, consider the data frame `mtcars`, which is available in R

```
> data(mtcars)
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Names of the columns in the data frame can be obtained using `names`. Row names can be extracted using `rownames`.

```
> names(mtcars)
  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"
 [10] "gear" "carb"
> head(rownames(mtcars),5)
[1] "Mazda RX4"          "Mazda RX4 Wag"      "Datsun 710"
[4] "Hornet 4 Drive"     "Hornet Sportabout"
> # Getting the size of the data frame
> dim(mtcars)
[1] 32 11
```

Columns of the data frame can be extracted using a dollar sign.

```
> mtcars$mpg
  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3
 [14] 15.2 10.4 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3
 [27] 26.0 30.4 15.8 19.7 15.0 21.4
```

Elements and subsets can also be extracted using brackets. Both numbers and names can be used for identifying rows and columns.

```
> mtcars[2,3]
[1] 160
> mtcars["Mazda RX4 Wag","disp"]
[1] 160
> mtcars["Mazda RX4 Wag",]
              mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4 Wag  21   6  160 110  3.9 2.875 17.02  0  1    4    4
> mtcars[10:15,"hp"]
[1] 123 123 180 180 180 205
> class(mtcars[10:15,1:2])
[1] "data.frame"
```

Using data in data frames

Data contained in data frames can be used in several ways.

- Attaching the data frame. This includes the data frame in the search list, so the columns in it can be identified as variables.

```
> attach(mtcars)
> search()
 [1] ".GlobalEnv"        "mtcars"            "tools:rstudio"
 [4] "package:stats"     "package:graphics"  "package:grDevices"
 [7] "package:utils"     "package:datasets"  "package:methods"
[10] "Autoloads"         "package:base"
> objects(2)
 [1] "am"   "carb" "cyl"  "disp" "drat" "gear" "hp"   "mpg"
 [9] "qsec" "vs"   "wt"
> summary(mpg)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  10.40   15.42   19.20   20.09   22.80   33.90
```

Once the data frame is not needed anymore, it can be detached

```
> detach(mtcars)
> search()
 [1] ".GlobalEnv"         "tools:rstudio"     "package:stats"
 [4] "package:graphics"   "package:grDevices" "package:utils"
 [7] "package:datasets"   "package:methods"   "Autoloads"
[10] "package:base"
> summary(mpg)
Error in summary(mpg) : object 'mpg' not found
```

- Using `with`

```
> with(mtcars,summary(mpg))
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  10.40   15.42   19.20   20.09   22.80   33.90
```

Modelling functions have options for specifying the data frame where data are.

Manipulating data frames with dplyr

dplyr is a package in R which facilitates some data manipulation tasks.
We will present two basic commands: `filter` and `select`.
Suppose, for example, that we want to extract those cars with 4 cylinders and mechanical transmission. This can be done using `filter`

```
> library(dplyr)
> mech4c=filter(mtcars,cyl==4&am==1)
> View(mech4c)
> class(mech4c)
[1] "data.frame"
```

The result is a new data frame with eight rows.

Suppose now that we want to extract mpg for those cars (4 cylinders, mechanical transmission). Command `select` in `dplyr` makes this task easier.

```
> mpg.4cm=select(mech4c,mpg)
> View(mpg.4cm)
> class(mpg.4cm)
[1] "data.frame"
```

For transforming this data frame into a numerical vector, the command `unlist` in R can be used

```
> mpg.4cm=unlist(mpg.4cm)
> class(mpg.4cm)
[1] "numeric"
```

All this process can be done in one line of code using "piping", that is, sending the result of one command in R automatically to another command.
The "pipe" command in `dplyr` is %>%

```
> mph.4cm=filter(mtcars,cyl==4&am==1) %>% select(mpg) %>%
 unlist()
```

Working directory

R will read and write information from a *working directory*. Working directory can be changed during the session.

```
> # Get actual working directory
> getwd()
[1] "C:/Users/Maria Eglee/Dropbox/Cursos/MATE3026/Agosto 2014/Evaluaciones"
> # Set new working directory
> setwd("C:/Users/Maria Eglee//Dropbox//Grants//U54//R workshop//Slides")
> getwd()
[1] "C:/Users/Maria Eglee/Dropbox/Grants/U54/R workshop/Slides"
```

Working directory can also be set using menus, both in R and in RStudio

Reading data

- Reading data from the console: The command `scan` can be used to read from the consoles (it can be used for reading files too, though we are not going to use it for that purpose).

```
> x=scan()
1: 2 4 6 3 6 2
7: 1 3 5
10: 2 3
12:
Read 11 items
> x
 [1] 2 4 6 3 6 2 1 3 5 2 3
```

```
> y=scan(,what=list(0,""))
1: 1 A
2: 2 B
3: 3 D
4: 4 5
5:
Read 4 records

> class(y)
[1] "list"
```

- Reading delimited files:
  Text files can be read using `read.table`.
  The default separator for `read.table` is a "blank space" (any number of spaces or tabs).
  We will use a dataset included in the Agresti and Franklin (2012) CD, containing data on human development (Program of the United Nations for Development, PNUD)

```
> HumanDevelopTxt=read.table("./Datasets/human_development.txt",
+ header=T,sep="")
 Error in scan(file, what, nmax, sep, dec, quote, skip, nlines,
 na.strings,  : line 24 did not have 7 elements
```

  This happens because some country names have two or more words!

To solve this, the separator can be substituted by a tab.

```
> HumanDevelopTxt=read.table("./Datasets//human_development.txt",
+ header=T,sep="\t")
> class(HumanDevelopTxt)
[1] "data.frame"
> head(HumanDevelopTxt,5)
       C1.T INTERNET   GDP  CO2 CELLULAR FERTILITY LITERACY
1   Algeria     0.65  6.09  3.0      0.3       2.8     58.3
2 Argentina    10.08 11.32  3.8     19.3       2.4     96.9
3 Australia    37.14 25.37 18.2     57.4       1.7    100.0
4   Austria    38.70 26.73  7.6     81.7       1.3    100.0
5   Belgium    31.04 25.52 10.2     74.7       1.7    100.0

> HumanDevelopTxt[24,]
          C1.T INTERNET   GDP CO2 CELLULAR FERTILITY LITERACY
24 New Zealand    46.12 19.16 8.1     59.9         2      100
```

One of the variants of `read.table`, `read.delim`, already includes this change and expects column names by default.

```
> HumanDevelopTxt=read.delim("./Datasets/human_development.txt")
> head(HumanDevelopTxt)
       C1.T INTERNET   GDP  CO2 CELLULAR FERTILITY LITERACY
1   Algeria     0.65  6.09  3.0      0.3       2.8     58.3
2 Argentina    10.08 11.32  3.8     19.3       2.4     96.9
3 Australia    37.14 25.37 18.2     57.4       1.7    100.0
4   Austria    38.70 26.73  7.6     81.7       1.3    100.0
5   Belgium    31.04 25.52 10.2     74.7       1.7    100.0
6    Brazil     4.66  7.36  1.8     16.7       2.2     87.2
```

Another variant of `read.table`, `read.csv`, allows to read comma separated values files (which can be produced from Excel)
This is usually the easiest way of getting data into R (personal opinion!).

```
> HumanDevelopCsv=read.csv("./Datasets/human_development.csv")
> head(HumanDevelopCsv,5)
       C1.T INTERNET   GDP  CO2 CELLULAR FERTILITY LITERACY
1   Algeria     0.65  6.09  3.0      0.3       2.8     58.3
2 Argentina    10.08 11.32  3.8     19.3       2.4     96.9
3 Australia    37.14 25.37 18.2     57.4       1.7    100.0
4   Austria    38.70 26.73  7.6     81.7       1.3    100.0
5   Belgium    31.04 25.52 10.2     74.7       1.7    100.0
```

- Reading files from other statistical software:
  The package `foreign` in R contains functions for reading and writing data stored by some versions of Minitab, SAS, SPSS, Stata (frozen at version 12, but your can use `saveold` is Stata) and other statistical software
  For example, consider a SPSS version of the human development dataset

```
> require(foreign)
> HumanDevelopSpss=read.spss("./Datasets/human_development.sav",
+ to.data.frame=T)
> head(HumanDevelopSpss,5)
             C1T INTERNET   GDP  CO2 CELLULAR FERTILITY LITERACY
1 Algeria               0.65  6.09  3.0      0.3       2.8     58.3
2 Argentina            10.08 11.32  3.8     19.3       2.4     96.9
3 Australia            37.14 25.37 18.2     57.4       1.7    100.0
4 Austria              38.70 26.73  7.6     81.7       1.3    100.0
5 Belgium              31.04 25.52 10.2     74.7       1.7    100.0
```